Elmar Schömer
Ann-Christin Wörl

10. Übungsblatt

Abgabe: Dienstag, der 16.01.2024, 14:00 Uhr

Aufgabe 1: Klammerung (5+10+10 Punkte)

Wir wollen überprüfen, ob eine Zeichenfolge, die nur aus öffnenden und schließenden Klammern besteht, korrekt geklammert ist (siehe auch https://de.wikipedia.org/wiki/Dyck-Sprache). Wir betrachten im Folgenden nur Zeichenfolgen, die aus den Zeichen '(',')','[' und ']' aufgebaut sind.
Korrekt geklammerte Zeichenketten können wie folgt gebildet werden:

1. Die leere Zeichenfolge ε =" ist korrekt geklammert.
2. Wenn u und v zwei korrekt geklammerte Zeichenfolgen sind, so ist auch deren Aneinanderreihung (Konkatenation) $u\,v$ korrekt geklammert.
3. Wenn u korrekt geklammert ist, so sind auch (u) und $[u]$ korrekt geklammert.

Beispiele: '([][](()))' ist korrekt geklammert. '())(([' ist nicht korrekt geklammert.
Der folgende Pseudocode dient zur Erkennung korrekt geklammerter Zeichenketten. Er benutzt als Datenstruktur einen Stapelspeicher. Als einen Stapelspeicher (engl. stack) (siehe https://de.wikipedia.org/wiki/Stapelspeicher) bezeichnet man eine abstrakte Datenstruktur, die nach dem Last-In-First-Out-Prinzip (LIFO) funktioniert. Das heißt, Elemente können nur oben auf den Stapel gelegt und auch nur von dort wieder gelesen werden. Elemente werden übereinander gestapelt und in umgekehrter Reihenfolge vom Stapel genommen.

```
1  def ist_korrekt_geklammert(w):
2      S = leerer Stapel
3      for i in range(len(w)) do
4          if w_i öffnende Klammer then
5              lege w_i oben auf den Stapel S
6              continue
7          if w_i schließende Klammer then
8              if Stapel S leer then
9                  return False
10             if w_i korrespondiert nicht zu oberstem Stapelelement then
11                 return False
12         lösche oberstes Stapelelement
13     if Stapel S leer then
14         return True
15     else
16         return False
```

1. Erklären Sie, warum dieser Algorithmus erkennen kann, ob ein Klammerausdruck korrekt geklammert ist.
2. Entwerfen Sie eine Klasse `Stapel`, die alle nötigen Operationen zur Implementierung der Funktion `ist_korrekt_geklammert` unterstützt.
3. Implementieren Sie die Funktion `ist_korrekt_geklammert`.

Aufgabe 2: Vektoren (5+10+5+10 Punkte)

Wir wollen eine Klasse `Vektor` entwickeln, mit der wir die typischen Vektoroperationen im $\mathbb{R}^n$ ausführen können. Seien $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ und $s \in \mathbb{R}$, dann können wir folgende komponentenweisen Operationen definieren:
Addition und Subtraktion:

$$\mathbf{a} \pm \mathbf{b} = (a_0, a_1, \ldots, a_{n-1}) \pm (b_0, b_1, \ldots, b_{n-1}) = (a_0 \pm b_0, a_1 \pm b_1, \ldots, a_{n-1} \pm b_{n-1}) \in \mathbb{R}^n$$

Multiplikation mit einem Skalar:

$$\mathbf{a} \cdot s = (a_0, a_1, \ldots, a_{n-1}) \cdot s = (a_0 \cdot s, a_1 \cdot s, \ldots, a_{n-1} \cdot s) \in \mathbb{R}^n$$

Division durch einen Skalar:

$$\mathbf{a}/s = (a_0, a_1, \ldots, a_{n-1})/s = (a_0/s, a_1/s, \ldots, a_{n-1}/s) \in \mathbb{R}^n$$

Skalarprodukt:

$$< \mathbf{a}, \mathbf{b} >= \sum_{i=0}^{n-1} a_i \cdot b_i \in \mathbb{R}$$

1. Die Klasse `Vektor` besitzt schon einen Konstruktor. Recherchieren Sie die Bedeutung des `*`-Operators bei der Argumentübergabe.

2. Implementieren Sie die oben genannten Operationen für Vektoren, sodass die vektoriellen Ausdrücke im Hauptprogramm ausgewertet werden können.

3. Recherchieren Sie, welche Build-in Funktionen überladen werden müssen, um die letzten beiden Programmzeilen ausführen zu können.

4. Wir wollen nun nicht mehr mit Fließkomma-Arithmetik, sondern mit rationaler Arithmetik arbeiten. Wir betrachten also Vektoren im $\mathbb{Q}^n$. Passen Sie Ihre Vektor-Klasse so an, dass man anstelle von `float`- oder `int`-Werten auch Brüche verwenden kann.

```
1  class Vektor:
2      def __init__(self,*v):
3          self.v = []
4          for x in v:
5              self.v.append(x)
6
7  L = (1,2,3)
8  a = Vektor(*L)
9  b = Vektor(4,5,6)
10
11 print(a+b)
12 print(Vektor.Skalarprodukt(a/4+b*3,a*2+b/5))
13
14 a[1] = 0
15 print(a[1])
```

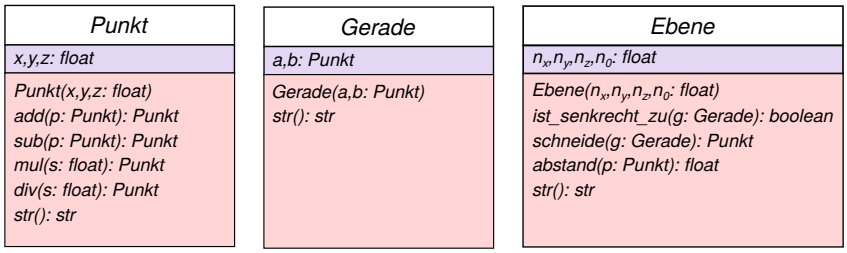
Aufgabe 3: Punkte, Geraden, Ebenen (20+10* Punkte)

Folgende Geometrie-Aufgabe stammt aus dem Mathe-Abitur Bayern (2016):

Gegeben sind die Ebene $E: 2x + y + 2z = 6$ sowie die Punkte $P = (1, 0, 2)$ und $Q = (5, 2, 6)$.

1. Zeigen Sie, dass die Gerade durch die Punkte P und Q senkrecht zur Ebene E verläuft.
2. Die Punkte P und Q liegen symmetrisch zu einer Ebene F. Ermitteln Sie eine Gleichung von F.

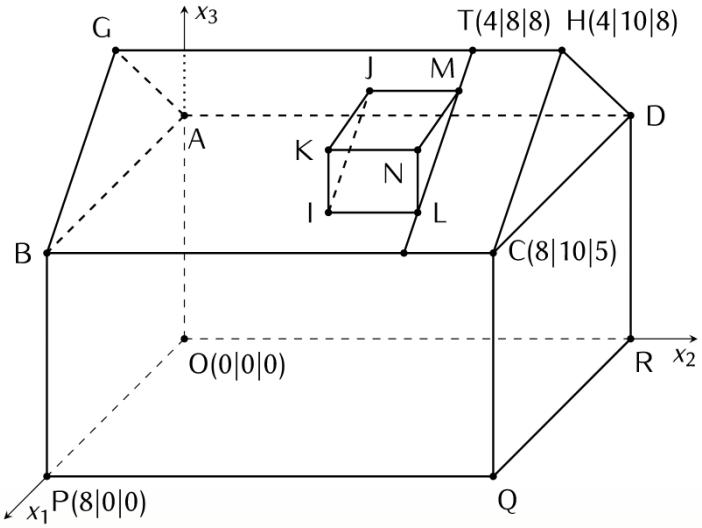
Schreiben Sie drei Klassen `Punkt`, `Gerade` und `Ebene` nach den (Mindest-)Vorgaben der folgenden Klassendiagramme, so dass Sie die obige Geometrie-Aufgabe nach dem Vorbild des gegebenen Python-Programms lösen können.




```python
import math

def Skalarprodukt(a,b):
    return(a.x*b.x+a.y*b.y+a.z*b.z)

class Punkt:
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z

class Gerade:

class Ebene:

P = Punkt(1,0,2)
Q = Punkt(5,2,6)
g = Gerade(P,Q)
E = Ebene(2,1,2,6)

print(E.ist_senkrecht_zu(g))

M = (P+Q)/2
print(M)
normale = Q-P
d = Skalarprodukt(Q-P,M)

F = Ebene(normale.x,normale.y,normale.z,d)
print(F)
print(F.abstand(P),F.abstand(Q))
```

Zusatzaufgabe:
Erweitern Sie die Funktionalität Ihrer Klassen, so dass Sie auch folgende Geometrie-Aufgabe aus dem Mathe-Abitur Bayern (2014) lösen können.

Die Abbildung zeigt modellhaft ein Einfamilienhaus, das auf einer horizontalen Fläche steht. Auf einer der beiden rechteckigen Dachflächen soll eine Dachgaube errichtet werden. Die Punkte A, B, C, D, O, P, Q und R sind die Eckpunkte eines Quaders. Das gerade dreiseitige Prisma $LMNIJK$ stellt die Dachgaube dar, die Strecke GH den First des Dachs, d.h. die obere waagrechte Dachkante. Eine Längeneinheit im Koordinatensystem entspricht $1m$, d.h. das Haus ist $10m$ lang.



1. Berechnen Sie den Inhalt derjenigen Dachfläche, die im Modell durch das Rechteck $BCHG$ dargestellt wird.

2. In der Stadt, in der das Einfamilienhaus steht, gilt für die Errichtung von Dachgauben eine Satzung, die jeder Bauherr einhalten muss. Diese Satzung lässt die Errichtung einer Dachgaube zu, wenn die Größe des Neigungswinkels der Dachfläche des jeweiligen Hausdachs gegen die Horizontale mindestens $35°$ beträgt. Zeigen Sie rechnerisch, dass für das betrachtete Einfamilienhaus die Errichtung einer Dachgaube zulässig ist.

   Die Dachfläche, auf der die Dachgaube errichtet wird, liegt im Modell in der Ebene $E : 3x_1 + 4x_3 - 44 = 0$.

   Die Dachgaube soll so errichtet werden, dass sie von dem seitlichen Rand der Dachfläche, der im Modell durch die Strecke HC dargestellt wird, den Abstand $2m$ und vom First des Dachs den Abstand $1m$ hat. Zur Ermittlung der Koordinaten des Punkts M wird die durch den Punkt $T(4, 8, 8)$ verlaufende Gerade

$$t : \mathbf{x} = \begin{pmatrix} 4 \\ 8 \\ 8 \end{pmatrix} + \lambda \cdot \begin{pmatrix} 4 \\ 0 \\ -3 \end{pmatrix},\ \lambda \in \mathbb{R}$$

   betrachtet.

3. Begründen Sie, dass t in der Ebene E verläuft und von der Geraden HC den Abstand 2 besitzt.

4. Auf der Geraden t wird nun der Punkt M so festgelegt, dass der Abstand der Dachgaube vom First $1m$ beträgt. Bestimmen Sie die Koordinaten von M. Zwischenergebnis: $M(4.8, 8, 7.4)$

   Die Punkte M und N liegen auf der Geraden

$$m : \mathbf{x} = \begin{pmatrix} 4.8 \\ 8 \\ 7.4 \end{pmatrix} + \mu \cdot \begin{pmatrix} 6 \\ 0 \\ -1 \end{pmatrix},\ \mu \in \mathbb{R},$$

die im Modell die Neigung der Dachfläche der Gaube festlegt. Die zur x_3-Achse parallele Strecke NL stellt im Modell den sogenannten Gaubenstiel dar; dessen Länge soll $1.4m$ betragen. Um die Koordinaten von N und L zu bestimmen, wird die Ebene F betrachtet, die durch Verschiebung von E um $1.4m$ in positive x_3-Richtung entsteht.

5. Begründen Sie, dass $3x_1 + 4x_3 - 49.6 = 0$ eine Gleichung von F ist.

6. Bestimmen Sie die Koordinaten von N und L. Teilergebnis: $N(7.2, 8, 7)$

Aufgabe 4: Objektorientierte Breitensuche (25 Punkte)

Wir betrachten noch einmal die Wegesuche in einem Graphen (siehe 5. Vorlesung). Wir wollen den Code zur Breitensuche in eine objektorientierte Form überführen. Dazu wollen wir eine Klasse `Schlange` und eine Klasse `Graph` entwerfen.
Die Klasse `Schlange` soll dazu dienen, die abstrakte Datenstruktur einer Warteschlange zu realisieren (siehe https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)), die nach dem Prinzip First-In-First-Out (FIFO) funktioniert. Die Funktionalität dieser Klasse soll von der Breitensuche verwendet werden.
Die Klasse `Graph` soll folgende Dinge implementieren:

- einen Konstruktor, um einen Graphen von einer Datei einzulesen
- eine Memberfunktion `ausgabe_der_Adjazenzlisten` zur Ausgabe der Adjazenzlisten des Graphen
- eine Memberfunktion `breitenSuche`, um von einem Startknoten aus eine Breitensuche zu starten
- eine Memberfunktion `ausgabe_des_kuerzesten_Weges`, um vom Startknoten einen kuerzesten Weg zu einem Zielkonten zu berechnen

```
1  class Schlange:
2      def __init__(self):
3
4      def hinten_an_die_Schlange_anfuegen(self,e):
5
6      def vorderstes_Element_der_Schlange(self):
7
8      def vorderstes_Element_der_Schlange_ausgeben_und_entfernen(self):
9
10     def ist_Schlange_leer(self):
11
12 class Graph:
13     def __init__(self,dateiname):
14
15     def ausgabe_der_Adjazenzlisten(self):
16
17     def breitenSuche(self,startKnoten):
18
19     def ausgabe_des_kuerzesten_Weges(self,startKnoten,zielKnoten):
20
21 G = Graph('huepfburg0.txt')
22 G.ausgabe_der_Adjazenzlisten()
23 G.breitenSuche(1)
24 G.ausgabe_des_kuerzesten_Weges(1,2)
```